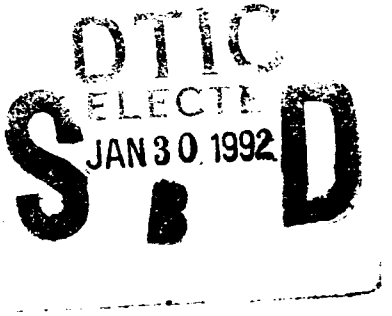

AFIT/EN-TR-91-9

Air Force Institute of Technology

Ada Interfaces to the X Window System

Gary W. Klabunde
Capt, USAF

Mark A. Roth
Maj, USAF

30 December 1991

Ada Interfaces to the X Window System

Gary W. Klabunde
Mark A. Roth

December 30, 1991

1 Introduction

The user interface is the component of the application through which the user's actions are translated into one or more requests for services of the applications, and that provides feedback concerning the outcome of the requested actions [7]. Because of the importance of this interaction, the design of efficient and easy to use user interfaces is receiving increased attention. Most people now realize that if an application has a user interface that is "unfriendly" or difficult to use, it is probably going to sit on the shelf unused. Also, user interfaces using some type of windowing system are fast becoming a common feature of most computer systems. As a result, users tend to expect all application programs to have a professional, polished user-friendly interface [11]. Unfortunately, the Ada language has only rudimentary input and output (I/O) capabilities. As such, user interface programmers using Ada had to develop some other methods for anything except simple line or character I/O.

This is not the case, however, for some other programming languages. The introduction of the X Window System in the mid 1980's changed the way user interfaces were developed in the language C. The X Window System, or X, is a collection (library) of subroutines that allows for the creation and manipulation of graphical user interfaces using multiple windows. These subroutines provide the mechanism to achieve the goals previously discussed.

Recognizing the importance of X to the development of user interfaces, some members of the Ada community began working on ways access the X Window System from within Ada programs. The first efforts involved developing bindings to the X routines. Subsequent efforts have looked at ways to implement X in the Ada language.

This paper discusses some of the more significant accomplishments in accessing X from Ada programs. Particular attention is paid to the bindings developed by Stephen Hyland formerly of Science Applications International Corporation and E.J. Jones of Boeing Aerospace Corporation. A discussion is presented of how these bindings were successfully used at the Air Force Institute of Technology for the design and implementation of a user interface for the Saber computer wargame. The paper ends with a description of the limitations of the bindings.

2 Ada and the X Window System

Originally, Xlib, Xt Intrinsics and most widget sets were written in the programming language C. Until a few years ago, there was no way for an application program written in Ada to use the X Window System. Recent efforts have taken two approaches: Ada bindings to X and Ada implementations of the X libraries. Most of the Ada bindings are tied to particular operating systems and will only work with a particular Ada compiler. The Alsys, Meridian, and Verdix compilers, along with their derivatives, are used most often for the bindings [1].

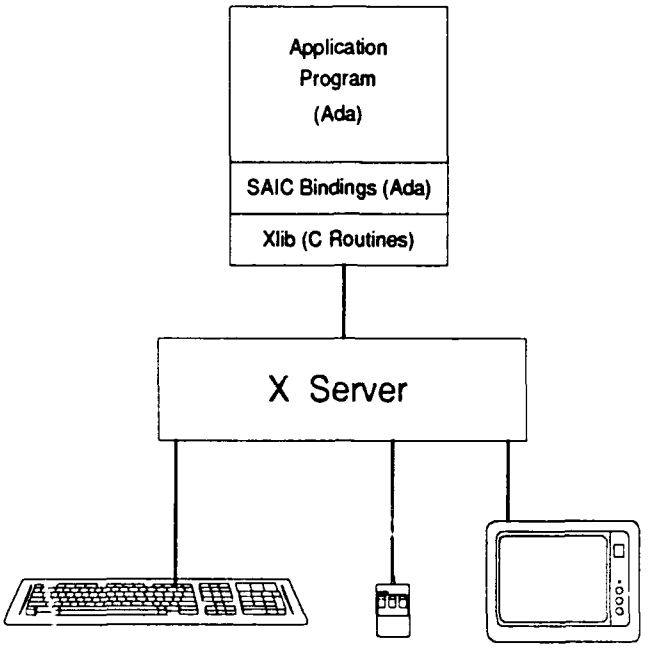

Figure 1: Application Program Configuration Using the SAIC Bindings

2.1 Ada Bindings to X

In 1987, the Science Applications International Corporation (SAIC) developed Ada bindings to the Xlib C routines. Their work was performed under a Software Technology for Adaptable Reliable Systems (STARS) Foundation contract, and is therefore in the public domain. According to Kurt Wallnau, "...a substantial effort was made to map the C data types to Ada, and do as much Xlib processing in Ada as possible before sending the actual request to the C implementation" [9:5]. The actual Ada interface is accomplished through the use of Ada *pragma interface* statements [3]. Put simply, the *pragma interface* construct allows an Ada program to call subprograms written in another language [2]. Figure 1 shows the configuration of an Ada program using the SAIC bindings to interface with Xlib. In this figure, the application program has no access to any toolkits or widget sets.

In a manner similar to that used by SAIC, the Boeing Corporation recently developed Ada bindings to a large subset of the Xt Intrinsics and the Motif widget set. Their code also provides access to a very limited subset of Xlib functions and data types. Like the SAIC code, Boeing's effort was sponsored by a STARS contract [5]. For the most part, the subroutine names and parameter lists closely mirror the actual C routines. Also, Boeing added a few subprograms to assist in the building of some commonly used parameter lists. The bindings require the Verdix Ada Development System (VADS) version 5.5 or higher to execute. While the documentation on the software is relatively sparse, it does indicate which modules would require changes in order to port the bindings to other systems.

Figure 2 shows the configuration of an Ada program using only the Boeing bindings. The dashed lines indicate that a small portion of the Xt Intrinsics and Motif functions are unavailable to the Ada program. Also, the application program cannot access the majority of the Xlib functions.

The Ada application program accesses the Xt Intrinsics and Motif routines by calling the appropriate subprogram in the bindings. For the most part, the bodies of the called subprograms contain code to convert the Boeing data structures and types to the types needed by the corresponding C code. The subprogram bodies then call internal procedures or functions that are bound to the Xt Intrinsics or Motif routines passing in the converted parameters.

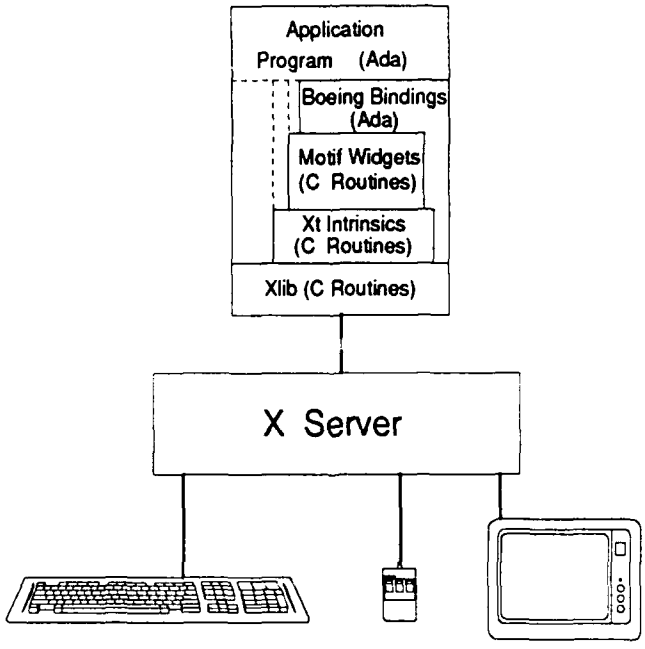

Figure 2: Application Program Configuration Using Boeing's Bindings

The bindings developed by Boeing and the SAIC are available at no cost to the Department of Defense. Recently, several other corporations have also developed bindings that are available for purchase [1]. These companies have basically taken one of two approaches. Some have followed the approach taken by the SAIC and Boeing. Others, such as Hewlett-Packard, took an alternative approach. To alleviate the need for much of the type conversion used by the SAIC and Boeing bindings, Hewlett-Packard binds the Ada subroutines directly to the corresponding C code. This results in very little code in the package bodies. To accomplish this, they make heavy use of Ada access types.

2.2 Ada Implementations

The USAF Electronic Systems Division recognized the need to write X Windows application programs in Ada at a higher level than through Xlib alone. In 1989, they sponsored a STARS Foundation contract to further research the capabilities of interfacing Ada and the X Window System [4]. The resulting reports documented efforts at integrating Ada with the X Toolkit (Xt).

As part of this STARS contract, Unisys Corporation developed an Ada implementation of (not bindings to) the X11R3 version of the Xt Intrinsics. "Ada/Xt," as it is called, "provides an intrinsics package which provides the functionality of Xt used to manage X resources, events and hierarchical widget construction" [10:1]. This software package uses a modified and corrected version of the SAIC bindings to interface to Xlib. Ada/Xt also includes a sample widget set consisting of ten Athena widgets and two HP widgets [10].

Unisys elected to develop an Xt implementation rather than Ada bindings, as SAIC did. The reasons for this included [9:9-10]:

1. The issue of widget extensibility. Ada bindings would require that new widgets be programmed in C.
2. The issues of inter-language runtime cooperation.
3. The issues of runtime environment interaction.

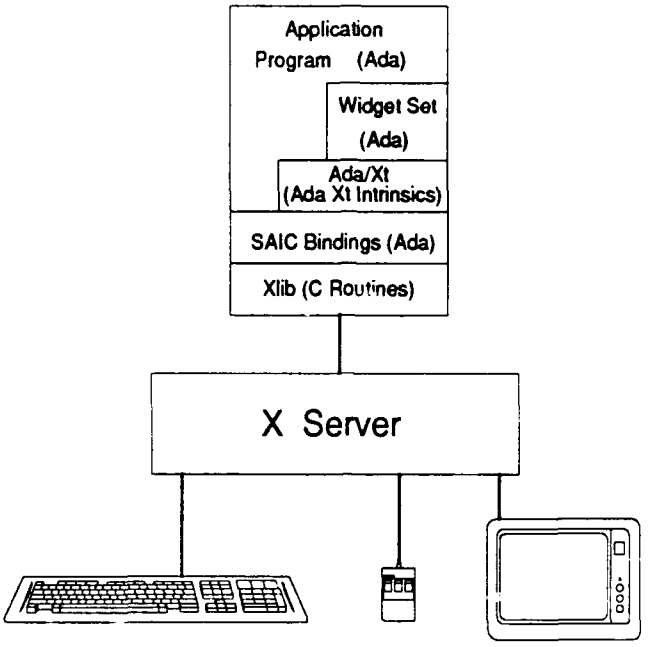

Figure 3: Application Program Configuration Using Unisys' Ada/Xt

Figure 3 represents a typical Ada application program using the Ada/Xt interface. The Ada application code can make use of the provided widgets, make calls to Ada/Xt, or make calls directly to the Xlib via the modified SAIC bindings. Thus, the full flexibility of an X application program written in C is maintained.

3 Saber: A Sample Application

Some user interfaces can be implemented by simply calling subroutines in the Xt Intrinsics and Motif widget set. Others may require additional calls to selected Xlib routines. The object-oriented graphical user interface for the Saber wargame [6] developed at the Air Force Institute of Technology fits into the latter category. Displaying the graphical symbols for the airbases, aircraft missions, and land units required the use of low-level Xlib subroutines.

Due to the need to access the Xlib, Xt Intrinsics and Motif libraries, it was clear that, as a minimum, the SAIC bindings would have to be used. The choice remained of whether to supplement it with the Boeing bindings or the Ada/Xt software developed by Unisys. Using the Ada/Xt software would have required the full or partial development of an Ada implementation of the Motif widget set. The Boeing software, on the other hand, already had bindings developed for Motif. Thus, we decided to utilize the Boeing bindings in combination with the SAIC software to develop the Saber user interface.

The Saber user interface was also designed to use a hexagon (hex) widget designed by the Air Force Wargaming Center (AFWC). This object-oriented widget contains routines to create and manipulate hexboards. Routines are provided to display certain features inside of a hex. These features include rivers, roads, cities, city names, forestation, and background color. Since the hex widget was written in the C programming language, Ada bindings had to be developed. These hex bindings were modeled after Boeing's bindings to the Motif widget set. Each procedure exported by the hex widget had to have a corresponding Ada procedure. To aid in understanding, the Ada procedure names were given the same names as their C counterparts except that underscores were inserted between words. Thus, the C procedure "Hx_SetHexLabel" became "Hx_Set_Hex_Label".

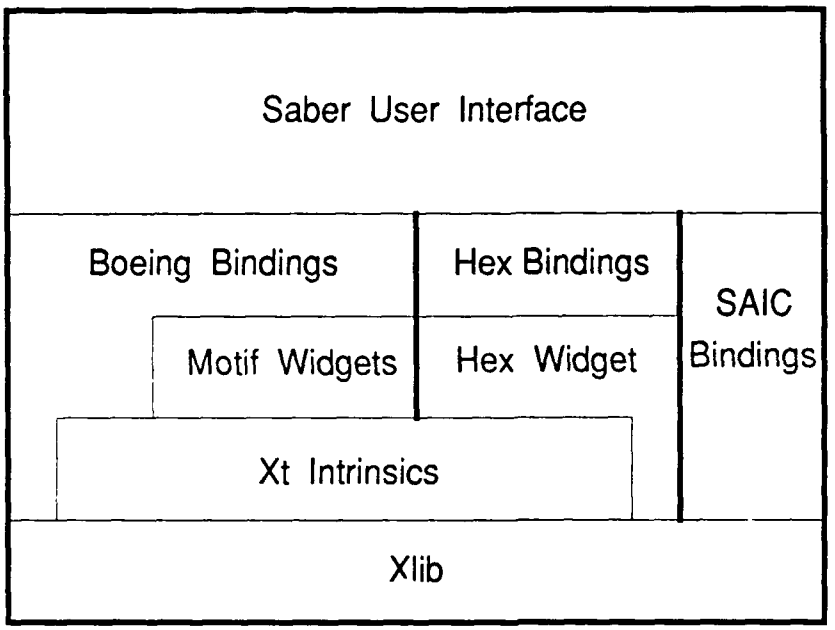

Figure 4: User Interface Relationship to the Ada Bindings

3.1 Combining the Ada Bindings

The relationship between the Saber user interface and the various Ada bindings is shown in Figure 4. This figure accurately reflects that the Boeing software contains bindings to a small subset of the Xlib functions in addition to the bindings to the Xt Intrinsics and Motif widget set. The user interface may make calls to the Boeing bindings, the SAIC bindings, and the hex widget bindings. In fact, interactions between the application program and the X Window System are made solely through these bindings.

The Boeing bindings were the primary means of interfacing with the X Window System, while the SAIC bindings were used primarily for the creation of the graphical unit symbols. Making the few calls to the SAIC bindings was not straightforward because of inconsistent types used by the two sets of bindings. Some inconsistencies were resolved by simple type conversion while others required the addition of new subroutines to the software.

3.1.1 Type Conversions.

By necessity, the Boeing software contains Ada declarations of a few low-level Xlib routines. These declarations for such things as the X Window System display, windows, and drawables were needed because the Xt Intrinsics provides functions to return these values that are created when the connection with the X server is established and windows are displayed on the screen.

Several of the SAIC procedures used to create the unit symbol pixmaps required these values as parameters. Two methods were used to convert the values to the types needed by the SAIC code. The first was a simple type conversion as in the following example that converts a float number to an integer:

```
integer_number := integer( float_number );
```

The second method used unchecked conversion, a predefined generic function provided as part of the Ada language. This generic function had to be instantiated with a source type and a target type for each conversion to be performed. An example instantiation to convert a variable of type "Display_Pointer"

returned by Boeing's *Xt_Display* function to a variable of type "Display" for use in the SAIC routines follows:

```
function Display_Id_From_Xt_Display is new Unchecked_Conversion
                          ( Source => XLIB.Display_Pointer,
                            Target => X_Windows.Display );
```

The unchecked conversion utility allows a sequence of bits, an address in the above example, to be treated as a variable of two different types. However, this capability should be used with caution. As Cohen writes, "Abuse of this capability can subvert the elaborate consistency-checking mechanisms built into the Ada language and lead to improper internal representations for data"[2:804]. For the Saber user interface, however, this was the only way to pass certain variables created through the Boeing bindings as input parameters to the SAIC subroutines.

3.1.2 Problems With SAIC Data Structures.

Since the initial connection with the X server was made through the Xt Intrinsics via the Boeing bindings, and not through the SAIC code, several internal SAIC data structures were not initialized. Because these data structures were not initialized, some functions provided by the SAIC bindings could not be used.

Two of the functions that fell into this category were *Default_Depth* and *Root_Window*. The results returned by these functions were needed for the creation of the unit symbol pixmaps. To obtain these values, a binding was created for each function and added to the Boeing bindings. Before the values could be used by the SAIC subroutines, however, they had to be converted to the corresponding SAIC types. The value returned by *Root_Window* was converted using the unchecked conversion described in the previous section, while the value returned by *Default_Depth* was converted through simple type conversion.

4 Limitations of the Bindings

The bindings written for Xlib, Xt Intrinsics and Motif widget set proved to be an indispensable part of the Saber user interface. While there were some weaknesses noted in the software, as a whole the bindings were able to directly or indirectly satisfy the requirements for the user interface. One problem common to the bindings is that they were designed for specific versions of the X software. Specifically, the SAIC bindings are for X11R3 and the Boeing bindings are for Motif V1.0.

4.1 Boeing Bindings

The first thing one notices when looking at the Boeing software is the lack of documentation. For the most part, the only documentation is in the form of section titles which separate the subroutines into topical categories. Thus, it would help if the application programmer is already familiar with the Xt Intrinsics and Motif widget set before trying to use the Boeing bindings. Furthermore, a few of the subroutines do not have nice, clean bindings to their corresponding C routines. These Ada subroutines use sparsely documented data structures that are defined within the bindings and that have no counterpart in the C code. It takes some time to learn what these data structures are for and how to use them properly.

A second weakness is that the bindings do not cover every Motif and Xt Intrinsics function. This fact is made clear in a "README" file that comes with the software. Some of the "missing" procedures can be added without too much difficulty. Other functions require a little more thought.

The third drawback to using the Boeing bindings is that they are currently tied to the Verdix Ada Development System (VADS) version 5.5 or higher. The bindings make use of the "C_Strings", "A_Strings", and "Command_Line" packages provided with the VADS library. The use of these packages restricts the portability of the application software. The "README" file included with the Boeing bindings indicates which modules would have to be changed to port the software to machines with different Ada compilers. However, the required changes should not be attempted by a novice Ada programmer.

4.1.1 Hardware Dependencies.

Even if your system does have VADS version 5.5 or higher, there is no guarantee that the Boeing bindings will work correctly. This fact was determined the hard way when attempting to use the bindings on a Sun 386i machine running VADS version 5.7 with Unix. Several test programs were written to gain familiarity with the bindings. However, they aborted with "Segmentation Faults" when executed. Analysis of the code showed that they were syntactically and semantically correct.

It was later determined that there were two problems, neither of which were caused by the Boeing bindings or the test programs. The causes of the problems were found in the August, 1991 edition of the *VADS Connection*. According to the Verdix Corporation, there are three potential problems areas to be aware of when writing programs that interface with C. These are parameter passing conventions, register usage, and parallelism. In this case, it was the first two areas that were causing the test program to abort.

The Verdix Corporation described the parameter passing conventions as follows[8:8]:

> In many cases, C does not use the same parameter passing conventions as Ada. When calling C from Ada this is not a problem, because VADS automatically generates a C calling sequence whenever pragma INTERFACE is used. When calling Ada from C, however, there can be a problem. Verdix has implemented pragma EXTERNAL, which will cause an Ada subprogram to accept a C calling sequence, but this is only available in version 6.0.5 and above.

The problem encountered with register usage had to do with differences in the ways Ada and C use registers. According to the Verdix Corporation[8:8]:

> For the 386...C expects the call to save and restore any registers it modifies, other than eax. Ada expects the caller to do the saving. This works fine when Ada calls C, but screws things up when C calls Ada. These register saves must be done manually, through the use of machine_code insertions.

At first glance, it did not appear that these issues would be causing the problems. It was obvious that Ada was making calls to C through the Boeing bindings, but it was not readily apparent that C was making any calls back to Ada. However, C was making calls to Ada inside of the *Xt_Main_Loop* procedure. Specifically, after the pushbutton is pressed, the C procedure *Xt_DispatchEvent* eventually causes control to be passed back to the Ada callback procedure that was registered with the pushbutton. It was at this point that the abovementioned problems caused the "Segmentation Fault".

However, it should be stressed that this was not a problem with the Boeing bindings. Rather, it is inherent in the way callback procedures are dispatched. The test programs and the Boeing bindings worked correctly when the software was executed on a Sun Sparc Station 2.

4.2 SAIC Bindings

A problem was also encountered with the SAIC bindings. These bindings were primarily used for the creation of the graphical symbols used to represent the air and land units in the Saber user interface. However, some problems were found when trying to read in the bitmap data created with the *Bitmap* editor provided with the X Window System software. This simple drawing program allows an application programmer to interactively create bitmap patterns. The pattern is saved in a special format that can be read in by an application program through calls to appropriate Xlib subroutines.

Analysis of the errors revealed that the SAIC programmers made a previous attempt to correct this problem. A solution to the problem was coded and tested that solved the problem without creating any new errors.

5 Conclusion

In this paper we have presented a brief overview of the X Window System along with recent efforts for incorporating its use into Ada programs. One method involving the use of Ada bindings to X was presented

in some detail. While there were a few exceptions, most of the Ada subprograms bear a close resemblance to their C counterparts. Thus, anyone familiar with the calling sequences for the Xlib, the Xt Intrinsics and the Motif widget set should be able to understand the functionality of Ada programs that use the Boeing and SAIC bindings. A detailed example of using these bindings can be found in [7].

The continued use of the SAIC and Boeing bindings is encouraged for the development of graphical user interfaces in Ada.

Acknowledgements

The research for this paper was supported by a grant from the Air Force Wargaming Center, AU CADRE/WG, Maxwell AFB, AL, 36112.

References

[1] Ada Information Clearinghouse. *Available Ada Bindings*. Draft. Lanham, MD, October 1991.

[2] Cohen, Norman H. *Ada as a Second Language*. New York: McGraw-Hill, 1986.

[3] Hyland, Stephen J. and Mark A. Nelson. "Ada Bindings to the X Window System." Ada computer software source code, 1987.

[4] *Interface Standards Informal Technical Data, Ada Interfaces to X Window System*. Software Technology for Adaptable Reliable Systems (STARS) Contract F19628-88-D-0031, Publication No. GR-7670-1069(NP), Reston VA: Unisys Corporation, March 1989 (AD-A228820).

[5] Jones, E. J. "Ada Bindings to the Xt Intrinsics and Motif Widget Set." Ada computer software source code, 1991.

[6] Klabunde, Capt Garv W. *An Animated Graphical Postprocessor for the Saber Wargame*. MS thesis, AFIT/GCS/ENG/91D-10, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

[7] Klabunde, Gary W. and Mark A. Roth. *Using Ada Bindings to the X Window System*. Technical Report AFIT/EN-TR-91-10, Wright-Patterson AFB OH: School of Engineering, Air Force Institute of Technology (AU), December 1991.

[8] Myers, Brad A. and Mary Beth Rosson. "User Interface Programming Survey," *SIGCHI Bulletin*, *23*:27–30 (April 1991).

[9] Verdix Corporation. *VADS Connection*. Technical Report. Chantilly, VA, August 1991.

[10] Wallnau, Kurt C. *Ada/Xt Architecture: Design Report*. Software Technology for Adaptable Reliable Systems (STARS) Contract F19628-88-D-0031, Publication No. GR-7670-1107(NP), Reston VA: Unisys Corporation, January 1990 (AD-A228827).

[11] Wallnau, Kurt C. and others. *Ada/Xt Toolkit, Version Description Document*. Software Technology for Adaptable Reliable Systems (STARS) Contract F19628-88-D-0031, Publication No. GR-7670-1133(NP), Reston VA: Unisys Corporation, July 1990 (AD-A229637).

[12] Young, Douglas. *The X Window System: Programming and Applications with Xt (OSF/Motif Edition)*. Englewood Cliffs NJ: Prentice Hall, 1990.

Author Information

GARY W. KLABUNDE is a systems analyst/programmer with 12 months experience in writing Ada user interfaces using the X Window System. MARK A. ROTH is an associate professor of computer systems in the Department of Electrical and Computer Engineering at the Air Force Institute of Technology. His current research interests include wargaming simulation and database management systems. Both are currently on active duty for the United States Air Force. Current addresses:

Capt Gary W. Klabunde
SCCC/SOSW
Offut AFB, NE

Maj Mark A. Roth
AFIT/ENG
Wright-Patterson AFB, OH 45433-6583

mroth@afit.af.mil

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	30 December 1991	Technical Report

4. TITLE AND SUBTITLE
Ada Interfaces to the X Window System

5. FUNDING NUMBERS

6. AUTHOR(S)
Gary W. Klabunde, Capt, USAF
Mark A. Roth, Maj, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/EN-TR-91-9

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Wargaming Center
AU CADRE/WG
Maxwell AFB AL, 36112-5532

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This paper discusses some of the more significant accomplishments in accessing X from Ada programs. Particular attention is paid to the bindings developed by Stephen Hyland formerly of Science Applications International Corporation and E.J. Jones of Boeing Aerospace Corporation. A discussion is presented of how these bindings were successfully used at the Air Force Institute of Technology for the design and implementation of a user interface for the Saber computer wargame. The paper ends with a description of the limitations of the bindings.

14. SUBJECT TERMS
Ada, X Windows, Motif, Ada Bindings

15. NUMBER OF PAGES
10

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL